Extended Abstract - Semantic query languages for knowledge-based web services in a construction context

Jeroen Werbrouck[a,b], Madhumitha Senthilvel[a], Jakob Beetz[a], Pierre Bourreau[c], Léon van Berlo[d]

[a] RWTH Aachen University, Germany
[b] Ghent University, Belgium
[c] Institut pour la Transition Energétique Nobatek/Inef4, France
[d] Netherlands Organisation for Applied Scientific Research TNO, the Netherlands

werbrouck@caad.arch.rwth-aachen.de

1. Introduction

Building Information Modelling (BIM) has proven itself an extremely valuable asset for the construction industry, in terms of design, planning, rule checking and collaboration. However, a lot of fragmentation persists; each of the many stakeholders involved in a project has different methods and different software tools that need to exchange information. Conversion to standard exchange formats such as the Industry Foundation Classes (IFC) enables such cross-domain communication only partly.

Over recent years, the use of web technologies in the building industry has been gaining momentum, considered promising mechanisms for reaching a more interoperable BIM practice. Specifically, this relates to (1) cloud-based applications, and (2) Linked Data and Semantic Web technologies. Cloud-based initiatives for streamlining collaboration in building projects have known a long history. Spanning nearly two decades, they go from the (discontinued) frameworks in the early 2000s IMSvr (Hietanen, 2002) and BLIS/SABLE (BLIS-project, 2002; Kiviniemi et al., 2005) to the more recent and popular model server BIMserver (Beetz et al., 2010), and the recently introduced concept of "BIM bots" (van Berlo, n.d.). These projects are all based on collaboration and exchange through IFC. However, IFC only spans certain disciplines and offers limited interoperability with domains that are not covered in the schema. Such interoperability can be provided by adopting Linked Data technologies.

The idea that the application of Linked Data concepts could seriously enhance interoperability and collaboration has been documented multiple times already (Beetz et al., 2005; Pauwels et al., 2017; Pauwels and Terkaj, 2016; Rasmussen et al., 2017), and several research projects are currently working on making Linked Data-based BIM more mature. With the use of the Linked Data standard RDF (Resource Description Framework)[1], it is possible to construct dedicated vocabularies for each related sub-discipline, connected with one another in a neutral format and without information loss. An additional advantage of such Linked Data vocabularies over traditional, monolithic, domain-specific information models such as IFC is their modularity, which increases the flexibility to address domain specific and local challenges.

In order to combine both topics, the threshold for developing Linked Data applications needs to be lowered (Verborgh, 2018). Web applications need to communicate in an efficient way, to enable the interoperability and automation potential given by these technologies. The W3C standard for creating, querying and updating Linked Data databases is SPARQL[2]. SPARQL has proven to be a very powerful querying language, but it is also known to be cumbersome

[1] https://www.w3.org/RDF/, W3C
[2] https://www.w3.org/TR/sparql11-query/, W3C

and a threshold for developers that do not have a lot of experience with Linked Data triples. To overcome these issues, initiatives such as GraphQL-LD (Taelman et al., 2018) and HypergraphQL[3] rely on the API querying language GraphQL[4], but extend it with Linked Data querying functionality, so it is possible to fetch the necessary data from the distributed RDF building model. The advantage of these initiatives is that no real Linked Data knowledge is needed to develop a service that makes nevertheless use of its core principles: GraphQL is flexible and clearly structured, designed to build client applications, their data requirements and interactions; the query response being structured according to the widely used JSON-format.

2. Methodology

In this paper, we investigate the use of existing open-source GraphQL-based solutions for querying the semantic web; HyperGraphQL and GraphQL-LD. Firstly, an overview is given about some past and present frameworks for cloud-based collaboration in construction, based on IFC. Then, recent developments regarding Linked Data for the building industry are outlined, followed by a brief discussion on Semantic Web query languages. A comparison is made between the solutions proposed by HyerGraphQL and GraphQL, according to different criteria, with a focus on schema versus schemaless deployment, the use of intermediary servers, type querying and the need for updating functionality. Whenever deemed relevant, example use cases are implemented for clarification and comparison.

3. Outcomes

This comparison results in a table that compares the discussed GraphQL approaches with one another and with SPARQL. Because different use cases are possible, the application of a query language can differ depending on scenario: communication between web-applications and databases is different from retrieving information from fixed datasets. Because GraphQL-LD uses a serverless and schemaless deployment, it is considered more flexible to implement in web applications. However, for querying specific datasets for specific data patterns, HyperGraphQL offers a good way of exposing certain data and hide other. The outcomes of this comparison will serve as an input for further development of Linked Data based solutions for collaboration in building projects, to come a step closer to the use of interoperable web-tools in practice.

4. Acknowledgements

The authors would like to acknowledge the support by both the BIM4REN project, which has received funding from the European Union's Horizon 2020 Research and Innovation Program under grant agreement No 820773 and the UGent Special Research Fund (BOF).

References

Beetz, J., van Berlo, L., de Laat, R., van den Helm, P., 2010. BIMserver. org–An open source IFC model server, in: Proceedings of the CIP W78 Conference. p. 8.

[3] https://www.hypergraphql.org/, Semantic Integration Ltd.
[4] https://facebook.github.io/graphql, Facebook Inc.

Beetz, J., Van Leeuwen, J.P., De Vries, B., 2005. An ontology web language notation of the industry foundation classes, in: Proceedings of the 22nd CIB W78 Conference on Information Technology in Construction. Technical University of Dresden, p. 670.

BLIS-project, 2002. BLIS Building Lifecycle Interoperable Software. URL http://www.blis-project.org/index2.html.

Hietanen, J., 2002. BLIS review: IMSvr. BLIS.

Kiviniemi, A., Fischer, M., Bazjanac, V., 2005. Integration of multiple product models: Ifc model servers as a potential solution, in: Proc. of the 22nd CIB-W78 Conference on Information Technology in Construction.

Pauwels, P., Terkaj, W., 2016. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. Automation in Construction 63, 100–133.

Pauwels, P., Zhang, S., Lee, Y.-C., 2017. Semantic Web Technologies in AEC Industry: A Literature Overview. Automation in Construction 73, 145–165.

Rasmussen, M., Pauwels, P., Lefrançois, M., Schneider, G.F., Hviid, C., Karlshøj, J., 2017. Recent Changes in the Building Topology Ontology, in: LDAC2017-5th Linked Data in Architecture and Construction Workshop.

Taelman, R., Vander Sande, M., Verborgh, R., 2018. GraphQL-LD: Linked Data Querying with GraphQL, in: ISWC2018, the 17th International Semantic Web Conference.

van Berlo, L., n.d. bimbots.org - Automate your work with BIM Bots. URL http://bimbots.org.

Verborgh, R., 2018. Designing a Linked Data Developer Experience. URL https://ruben.verborgh.org/blog/2018/12/28/designing-a-linked-data-developer-experience/.

Semantic query languages for knowledge-based web services in a construction context

Jeroen Werbrouck[a,b], Madhumitha Senthilvel[a], Jakob Beetz[a], Pierre Bourreau[c], Léon van Berlo[d]

[a] RWTH Aachen University, Germany
[b] Ghent University, Belgium
[c] Institut pour la Transition Energétique Nobatek/Inef4, France
[d] Netherlands Organisation for Applied Scientific Research TNO, the Netherlands

werbrouck@caad.arch.rwth-aachen.de

Abstract. Since the early 2000s, different frameworks were set up to enable web-based collaboration in building projects. Unfortunately, none of these initiatives was granted a long life. Recently, however, the use of web technologies in the building industry has been gaining momentum again, considered some promising technologies for reaching a more interoperable BIM practice. Specifically, this relates to (1) Linked Data and Semantic Web technologies, and (2) cloud-based applications. In order to combine these into a network of interlinked applications and datastores, an agreed-upon mechanism for automatic communication and data retrieval needs to be used. Apart from the W3C standard SPARQL, often considered too high a threshold for developers to implement, there are some recent GraphQL-based solutions that simplify the querying process and its implementation into web services. In this paper, we review two recent open source technologies based on GraphQL, that enable to query Linked Data on the web: GraphQL-LD and HyperGraphQL.

1. Introduction

Building Information Modelling (BIM) has proven itself an extremely valuable asset for the construction industry. However, a lot of fragmentation persists, due to the large number of stakeholders in a typical construction project, each one relying on specialized software packages that often use proprietary formats. This fragmentation makes it difficult to exploit the full potential of digitisation in the building industry. To improve interoperability, the Industry Foundation Classes (IFC)[1] are the international standard for exchange of construction-related data. However, because the IFC schema includes an enormous amount of classes and properties, software vendors almost never implement it completely; obviously, in most cases only the subsets that are relevant to the tool's purpose are considered. Additionally, the conversion process from native formats to IFC files is seldom watertight, making the ideal scenario of lossless information flow very difficult, if not impossible to achieve (Kiviniemi et al., 2005; Shafiq et al., 2018). Interoperability and interdisciplinarity become even more complicated when dealing with disciplines that are not taken into account in the IFC schema. These are mainly areas that relate to parts of the Building Life Cycle (BLC) that are not directly involved with construction itself, e.g. heritage conservation, Facility Management and implementation of GIS data (Geographic Information Systems).

To tackle at least some of these challenges, there have been a number of initiatives to make the construction industry more web-based, using web technologies for improved collaboration and interdisciplinary crossovers. Specifically, this relates to (1) the use of cloud-based, interconnected services (Afsari et al., 2016; Beetz et al., 2010b; Kiviniemi et al., 2005) and (2)

[1] http://www.buildingsmart-tech.org/specifications/ifc-overview

the use of Semantic Web technologies based on the Resource Description Framework (RDF)[2] (Beetz et al., 2005; Pauwels et al., 2017). Instead of 'monolithic' software suites that claim to encompass the entire BLC, cloud-based, service-oriented BIM applications dedicated to a single purpose are considered much more agile, since their modularity allows easy extension and custom chaining of tools. Recently, this service-oriented concept was re-introduced as 'BIM bots' (van Berlo, n.d.). As demonstrated by the recent Solid (Social Linked Data) framework (Mansour et al., 2016), these topics are tightly connected and do not only apply to the building industry, but to virtually all disciplines that are concerned with data exchange.

The combination of improvements in web infrastructure with current web technologies and their implementations for the building industry might provide the right receipt for the setup of a Linked Data-based descendant of the earlier frameworks such as IMSvr (Hietanen, 2002) BLIS/SABLE (BLIS-project, 2002; Kiviniemi et al., 2005) or the BIMserver (Beetz et al., 2010b). As in these frameworks, the most appropriate service for a specific task could be chosen out of a network of interconnected online tools. The main difference would be, however, that in this context the service fetches the data by requesting it from the distributed RDF building model rather than from an IFC file on a central server. Such approach is being actively pushed in the web community by initiatives such as Solid (Mansour et al., 2016) and OSLC (OASIS Open Project, n.d.). This way, not only BIM data from within the framework's software pool can be integrated, but also the immense stack of open data on the web, including non-BIM related information about geology, weather data, historic data etc.

For such a network of connected web services and data pods to exchange information efficiently and unambiguously, they need to make use of international standards. Therefore, an essential part for a successful set up of a Linked Data-based network of BLC-oriented services is to identify possible solutions for data exchange. A starting point is the W3C standard for querying Linked Data, SPARQL (SPARQL Protocol and RDF Query Language)[3]. However, SPARQL is often considered too high a threshold for software developers (Taelman et al., 2018b; Verborgh, 2018). Recently, a number of initiatives were launched for making Linked Data querying more accessible, by extending the GraphQL query language (Facebook Inc., 2016) to encompass Linked Data as well (Taelman et al., 2019): proprietary solutions such as Stardog and TopBraid, and open source technologies such as HyperGraphQL (Semantic Integration Ltd., n.d.) and GraphQL-LD (Taelman et al., 2018b). In this paper, we discuss the main differences between HyperGraphQL and GraphQL-LD, as these are open source solutions.

In the following sections, we first review existing concepts for cloud-based collaboration in construction projects, based on IFC (Section 2.1). This is followed by a brief introduction to Linked Data (Section 2.2), after which some characteristics of semantic web query languages are discussed (Section 2.3). Finally, they are compared according to multiple criteria, illustrated by some exemplary queries on a Linked Building Project (Section 3).

2. Related Work

2.1 IFC Model Servers

In the past, numerous methods have been explored for facilitating a collaborative platform for working with shared IFC models. File-based exchange between stakeholders gave rise to

[2] https://www.w3.org/RDF/, W3C, accessed 25/03/2019
[3] https://www.w3.org/TR/sparql11-query/, W3C, visited 15/12/2018

numerous problems including failure to handle IFC complexity and facilitate flexible data access. To combat these challenges, one of the potential solutions suggested was the use of Model Servers. Model Servers act as a database with a set of server applications that provide multi-user database management and also allow the use of IFC as the underlying database structure. A Model Server environment offers considerable advantages over file-based exchanges. For example, they provide flexibility for a multi-disciplinary management, scalability for larger projects, and more (Jørgensen et al., 2008).

Some notable initiatives in Model Server development are the BLIS/SABLE project, IMSvr, BIMServer, and BIMSie. SABLE aimed at increasing the interoperability application through semantic model sharing using SOAP. (BLIS-project, 2002). Its outputs were distinct domain specific APIs which handle the information exchange required by client applications for the respective domains. These relate to the BLIS views (Kiviniemi et al., 2005). On the other hand, in the IMSvr project, a combination of relational databases based on the IFC EXPRESS schema and a XML SOAP API was developed to expose server functionality. These initiatives were aimed at the entire organisation of BIM workflows through web technologies. However, real-world datasets proved much too complex and large for the web-technologies that were used back in the days; these were not able to extract and process multiple partial models for different construction-related tasks, with high computational demands, and requiring quick access to different data sources that contain disparate data. While these initiatives are now discontinued, the experiences gained in the projects and the results were constructive in developing future practices such as MVDs and IDM, which now form some of the basic concepts of IFC.

Two more recent Model Server developments, BIMServer and BIMSie, are currently being adopted by a wider community. BIMServer is based on a flexible model based layers including the model schema defined in ISO 10303 EXPRESS which in turn is used to generate a Meta Object Facility (MOF) described as an XML Metadata Interchange (XMI) model. Beyond these layers, BIMServer also facilitates user operations for management of data. Querying and filtering of data, user management and access rights are some of the possible operations in the current implementation (Beetz et al., 2010a). The BIMSie project furthered the BIMServer project through standardisation of a web services API for online BIM services. Such API standardisation enables machine-to-machine exchange of BIM data and stimulates innovation towards broader implementation of BIM bots. BIMSie is as protocol independent as possible and supports JSON and SOAP (van Berlo, 2013).

With the years, distributed, service-oriented architectures became more mature, allowing to establish chains of processes that connect different services over the globe. These chains of connected tools rely on advances in network technologies, which allow data exchange to take place at an unseen rate. At the same time, more and more sectors are adopting semantic web technologies (Berners-Lee et al., 2001), which allow to link individual statements (data at the smallest possible scale) with one another over the web. This happens in a way that this data can be uniquely identified all over the world, by both humans and digital actors, based on RDF.

2.2 Linked Data and the Building Industry

In the Linked Data technology stack, information is represented through the use of so-called triples, which resemble basic sentences with a subject, a verb (predicate) and an object. Each of these tree is denoted with a Uniform Resource Identifier (URI), which gives it a unique and universal context within the world wide web. This way, a data item can be referred to even if it is not stored on the same server, and can in turn refer to other data somewhere on the web, thus

actively contributing to a global graph of Linked Data. Furthermore, domain models can be developed with rules and taxonomies for certain disciplines. When distributed data makes use of such domain models ('ontologies' or 'vocabularies'), the 'semantic' in Semantic Web starts to make sense.

The idea that the application of Linked Data concepts could seriously enhance interoperability and collaboration in the AECO Industry (Architecture, Engineering, Construction, Operation) has been documented multiple times by now (Beetz et al., 2005; Pauwels et al., 2017; Pauwels and Terkaj, 2016; Rasmussen et al., 2017). With the use of Linked Data, it is possible to construct vocabularies for each related sub-discipline, connected with one another in a neutral format. This allows applications that are concerned with different aspects of the built environment to connect and exchange information without information loss. Furthermore, since vocabularies contain the information to interpret the data in a semantic way, automatic reasoning and rule checking comes within reach (Pauwels et al., 2017). With this in mind, several research projects are currently working on making Linked Data-based BIM more mature. The setup of a Linked Data-based version of IFC was first proposed in (Beetz et al., 2005) and has resulted in a buildingSMART-approved version ifcOWL (Pauwels and Terkaj, 2016). However, as ifcOWL covers the entire EXPRESS schema, it is a very large schema and therefore not flexible to deal with topics that are not covered in the IFC standard. The W3C Linked Building Data Working Group therefore targets the development of more modular vocabularies that each address a specific building-related topic (Rasmussen et al., 2017). The main advantage of such modular Linked Data vocabularies over traditional, monolithic, domain-specific information models such as the IFCs is their modularity, which increases the flexibility to address domain specific and local challenges.

2.3 Semantic Web Query Languages

As mentioned in Section 1, the W3C standard for creating, querying and updating Linked Data databases is SPARQL. SPARQL has proven to be a very powerful querying language, but it is also known to be cumbersome and a threshold for developers that have limited experience with Linked Data. To overcome these issues, initiatives such as GraphQL-LD and HypergraphQL rely on the API querying language GraphQL, but extend it with Linked Data querying functionality, making it possible to fetch the necessary data from the distributed RDF datasets by formulating simple GraphQL queries. The advantage of these initiatives is that no real Linked Data knowledge is needed to develop a service that makes nevertheless use of its core principles: GraphQL is flexible and clearly structured, designed to build client applications, data requirements and interactions.

'Plain' GraphQL is both a query language and a specification. It focuses on minimising and optimising data loads over the web, by connecting to a GraphQL API. Essential within GraphQL is a *schema* that describes *types*, their attributes (*fields*) and the relationships of those fields to other types or datatypes. In this perspective, the reference to graphs becomes obvious: there are nodes (the types) and edges (the fields) that point to other nodes. Note that the data structuring only happens in the schema, and therefore lacks the uniqueness of data and relationships in Linked Data graphs. Nevertheless, this structure allows to describe data in a linked and coherent way, and to query, update (*mutate*) and delete information.

As for developers, GraphQL as a query language is much more widely adopted than SPARQL. This is explained by the fact that the semantic web service stack is still quite small compared to services that use non-RDF databases, and that the learning curve for semantic web technologies such as SPARQL is still quite steep (Verborgh, 2018). In GraphQL, querying

happens in a more straightforward and 'cleaner' way: essentially, both GraphQL queries and their results are 'tree-like' queries, which make them easier to understand compared to SPARQL, which aims to query full graphs with a universal context. A comparison between a GraphQL query and a similar SPARQL query (with context) is given in table 1.

Table 1: GraphQL and SPARQL query

{
architect {
name
friends {
name
}
}
}
	PREFIX ex: <www.example.com/buildingteam#>
	PREFIX foaf: <http://xmlns.com/foaf/0.1/>
	
	SELECT ?name ?friendname
	WHERE {?person a ex:architect .
	?person foaf:name ?name .
	?person foaf:knows ?friend .
	?friend. foaf:name ?friendname .}

While there is essentially no relationship between GraphQL and SPARQL, a GraphQL query might be extended by some dictionary that relates the used concepts to a universal identification. This is exactly what happens in the semantic web extensions to GraphQL that will be discussed and compared in the next section: GraphQL-LD and HyperGraphQL.

3. Comparison

In this section, GraphQL-LD and HyperGraphQL are compared according to multiple criteria:

- Schema use
- Updating functionality
- Intermediary servers
- Federated querying
- Type querying
- Reverse querying

Whenever relevant, construction-related queries are used as illustrations. The sample dataset is a Linked Building Data (LBD) model of the CAAD chair at RWTH Aachen. This dataset was previously prepared for Linked Data queries by converting the original IFC file to its LBD equivalent, using the main LBD vocabularies BOT (Rasmussen et al., 2017), BuildingElement and PROPS. This conversion was done using the IFCtoLBD[4] converter (Bonduel et al., 2018).

3.1 GraphQL Schema

Semantic meaning in GraphQL is limited to the schema that is used. However, when extended with a mapping that relates the necessary definitions to a universal context, it can be used to also query RDF graphs. HyperGraphQL and GraphQL-LD implement this context differently.

The HyperGraphQL approach is closely related to the original GraphQL specifications. As with GraphQL, HyperGraphQL uses a schema that determines which data can be queried and which cannot. Each type and relationship in the schema is mapped to a URI that extends it to a universal context. Further on, both the type itself and its fields need to point to a specific service that needs to be queried.

[4] https://github.com/jyrkioraskari/IFCtoLBD, accessed 30/03/19

In contrast with HyperGraphQL, GraphQL-LD does not require a schema to be defined. In fact, it uses only the syntax of GraphQL to provide a streamlined querying experience, but 'under the hood', this query is translated to a SPARQL query, making use of a JSON context that maps the GraphQL definitions to URIs. The outcome of this GraphQL-to-SPARQL translation[5] is then used to perform a 'classic' SPARQL query. Results are presented in a tree-like structure[6], similar to a GraphQL response.

Both approaches have their benefits and disadvantages. Using a schema, HyperGraphQL provides a formal mechanism of projecting RDF data via a 'GraphQL lens', thereby affecting how clients can access certain data. On the one hand, this allows for fine-grained mapping of definitions to the services that are queried. On the other hand, this introduces some rigidness in the schema definition, as at the moment a class can only be mapped to one service[7]. The decision to only use a JSON-LD context and no schema enables a more flexible approach and allows to query the endpoint as a SPARQL query would, without restrictions on which data can be queried and which cannot. This means it also does not support (nor does it need) GraphQL schema introspection, which allows to see what kind of queries are supported: as all exposed data can be queried with GraphQL-LD, such introspection feature is superfluous.

3.2 Updating

Updating databases with GraphQL happens with so-called mutations. These allow to create, update and delete data. Similarly, SPARQL defines INSERT and DELETE; combining the two can be regarded as updating. This being said, at the moment of writing, neither GraphQL-LD nor HyperGraphQL supports mutations. When imagining a network of connected, interacting tools and online datastores owned by different stakeholders, updating is an essential prerequisite for enabling automatic communication. For now, regarding updating graph data on the web, developers are limited to either SPARQL queries or software libraries that hide SPARQL's complexity, offering simple functionality to manipulate data (e.g. rdflib, rdflib.js, LDflex).

3.3 Intermediaries

As mentioned in Section 3.1, the setup of HyperGraphQL requires the setup of an intermediary server, combining the provided schema and the Linked Data services defined in the configuration file to expose a GraphQL endpoint. The parameters for this intermediary service are given by a configuration file that points to the schema, the server port to use and the information required to identify the right services (graphs). Consequentially, when one needs to change the services to be queried, a new configuration file should be made, the schema types should be changed to point towards the new service, and one should run a new server.

GraphQL-LD, in contrast with the above approach, works serverless; as it is part of the Comunica framework (Taelman et al., 2018a), it can be embedded right into the application code or executed independently via the command line. This is quite an advantage in terms of ease-of-use for developers, and offers more flexibility for querying multiple services.

3.4 Data Federation

Both approaches allow to query multiple services. The main difference is that in HyperGraphQL, these services need to be defined in a configuration file before setting up the

[5] https://github.com/rubensworks/graphql-to-sparql.js, accessed 28/03/19
[6] https://github.com/rubensworks/sparqljson-to-tree.js, accessed 28/03/19
[7] https://www.hypergraphql.org/documentation/, accessed 30/03/19

intermediary server, while GraphQL-LD adds them dynamically before it carries out the query. Due to this implementation in Comunica, GraphQL-LD also allows to query graph data on the web that is not exposed through a SPARQL endpoint, such as regular Turtle graphs that are stored as documents. On the other hand, HyperGraphQL provides the functionality to query other, remote HyperGraphQL services on the web.

3.5 Querying by Type

An advantage of HyperGraphQL's usage of a schema, is that it automatically detects what can be queried as a type, and what is a property. This is then used by the intermediary server to automatically append to each type a 'GET' and a 'GET_BY_ID' extension, allowing to query items respectively by type (Fig. 1) or by URI (Fig. 2). A disadvantage is that a type should be stated explicitly in the graph, which is not always the case. For example, it cannot derive the results for *INST:Storey1 bot:hasSpace INST:Space1*, when in the graph it is not explicitly stated that *INST:Space1 a bot:Space*.

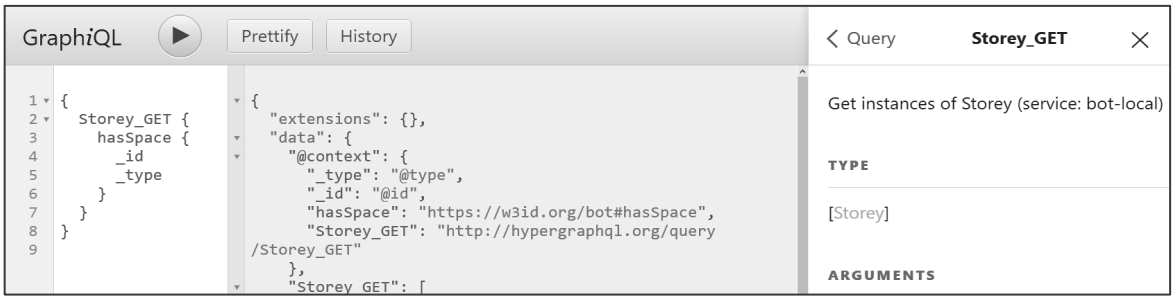

Figure 1: HyperGraphQL query to retrieve all instances of bot:Storey, by using Storey_GET. The introspection schema can be expanded in the GraphiQL interface.

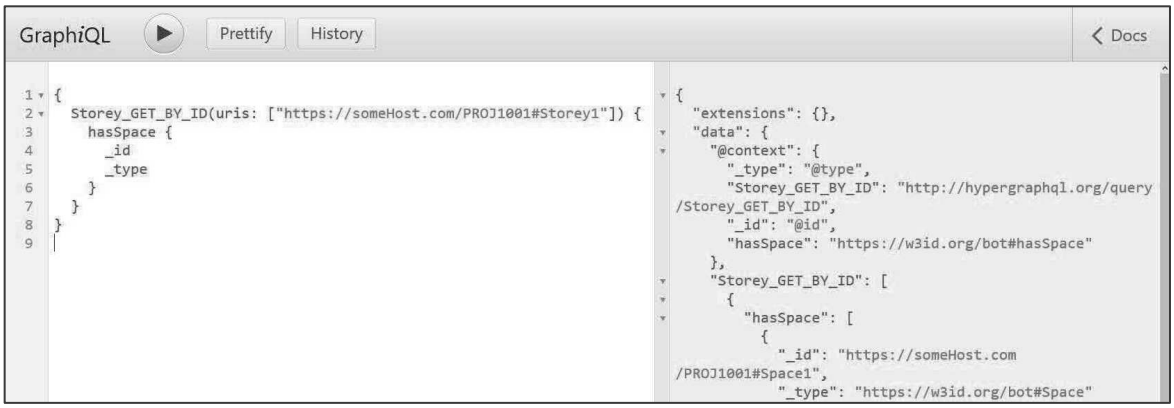

Figure 2: HyperGraphQL query to retrieve the instances of bot:Space of one specific instance of bot:Storey, by using Storey_GET_BY_ID.

Another point is that, as far as we know, it is currently not possible to map relationships to more than one schema type: for example, if the field 'hasSpace' is mapped to the type Space, we cannot make it apply to something which is an instance of type Kitchen, although this might be a subtype of Space. During the setup of our HyperGraphQL schema, any attempts to construct a GraphQL UNION type to solve this, failed. In RDF, on the contrary, a resource can be an instance of bot:Space and dogont:Kitchen at the same time. Since GraphQL-LD is in fact a translation of a GraphQL query to SPARQL, such multiple type assignments are no real issue.

GraphQL-LD is mainly focused on predicates, i.e. on querying by the relationships between nodes. Therefore, when querying for specific classes, a workaround is needed. The user should then include a '_' argument in the query and define 'a' in the context, referring to 'rdf:type'. The corresponding query is then *a(_:Storey)*. As an illustration, in Table 2, we define a more complex query for GraphQL-LD, that searches for the first 10 furniture elements of simple type "Stuhl (2): Stuhl (CIP-Pool)", and the bot:Space where this chair is located. The query contains limitation, type querying and value querying. Some tricks are necessary to get the URI of the furniture as well, since the translation from GraphQL-LD to SPARQL only takes the fields without curly braces as SPARQL variables. This results in a somewhat cumbersome 'double' query for 'containsElement'. This query has no equivalent in HyperGraphQL, since a 'reverse' context is used for backwards querying.

Table 2: Full GraphQL-LD query example

Context	{ "Element": "https://w3id.org/bot#Element", "Furniture": "https://w3id.org/product/Furnishing#Furniture", "containsElement": "https://w3id.org/bot#containsElement", "containedInZone": {"@reverse": "https://w3id.org/bot#containsElement"}, "simpleType": "https://w3id.org/props#type_simple", "a": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type", "simpleName": "https://w3id.org/props#nom_simple" }
Query	{ containsElement containsElement (first:10) { a(_:Furniture) simpleType(_:"Stuhl (2): Stuhl (CIP-Pool)") containedInZone{ simpleName } }}

SPARQL translation (prefixed)	SELECT ?containsElement ?containsElement_containedInZone_simpleName
	WHERE { ?b1 bot:containsElement ?containsElement.
	{SELECT * WHERE {
	?b1 bot:containsElement ?containsElement.
	?containsElement a furn:Furniture;
	props:type_simple "Stuhl (2): Stuhl (CIP-Pool)".
	?containsElement_containedInZone bot:containsElement ?containsElement;
	props:nom_simple ?containsElement_containedInZone_simpleName.
	} LIMIT 10 }
	}

Results	[0:{"containsElement":[{"value":"https://www.hackathon.org/caadchair#furniture_bb4d-87cf-d8a0"} {"containedInZone":[{"simpleName":[{"value":"Hiwis"}]}]}]} 1:{"containsElement":[{"value":"https://www.hackathon.org/caadchair#furniture_f879-4af7-88b5"} {"containedInZone":[{"simpleName":[{"value":"CIP-Pool"}]}]}]} 2:{...}]

The queries in Fig. 1, 2 and Table 2 also show that both approaches allow to define arguments in a query. While in HyperGraphQL, this might be used to specify the subject of the query itself (by the GET_BY_ID extension), GraphQL-LD is currently limited to relationships with objects.

3.6 Other

GraphQL-LD supports the use of an '@reverse' keyword in the JSON context (see Table 2). Practically, this means it allows to query both relationships 'hasSpace' and 'spaceOf. This functionality enables 'backwards querying', although the results will still be shown as a tree.

Note that this does not mean any reasoning is used to determine an 'InverseOf' relationship; this is supposed to be handled at the side of the SPARQL endpoint.

Similarities between both approaches are, amongs others, the functionality to specify limit and offset arguments, to set variables, to include aliases and set (inline) fragments.

3.7 Summary

In the above section, some large and small differences between two GraphQL-based approaches for querying linked data on the web were laid out. Table 3 shows the differences and similarities according to various parameters. SPARQL is also added to the comparison as a reference.

Table 3: Comparing different approaches for querying Linked Data on the Web

	GraphQL-LD	**HyperGraphQL**	**SPARQL**
Updating graphs	No	No	Yes
Schema and Introspection	No	Yes (+ introspection)	No
Intermediary server	No	Yes	No
Federated service querying	Yes	Yes	Yes
Reverse querying (O-P-S)	Yes	No	Yes

Both approaches have their application domain: using HyperGraphQL, querying fixed datasets (e.g. DBpedia.org or local multi-models) becomes more user friendly and uniform with a subset predefined set of types and fields. Therefore, we consider it mainly suitable for the setup of fixed endpoints to retrieve data from large, open databases.

On the other hand, GraphQL-LD seems to be more flexible regarding switching between endpoints, because of its serverless end schemaless deployment. If the goal is to simplify the development of Linked Data-based applications, the implementation of GraphQL-LD certainly takes less time and is more straightforward.

Because both initiatives have started rather recently, we hope the developers continue to work on their refinement, possibly implementing the option for applications to change the distributed datasets they communicate with, using a simple 'mutation'.

4. Conclusion

In this paper, an overview was given to some past and current frameworks for cloud-based collaboration in the AECO industry. Considering the upcoming of Linked Data and advancement in network technologies, the basic concepts of these early projects could be re-implemented in a Semantic Web based framework. Different solutions for communication between the services in such a framework and the distributed data these services use, were presented and compared. In context of a Linked Data based ecosystem, and in terms of a straightforward implementation, GraphQL-LD seems the most flexible choice.

Determining how tools within a framework can communicate is the first step in the development of a testing environment that consists of multiple modular services. The next steps will be the categorisation of such tools, by configuring a Linked Data 'use case ontology', that allows to express multiple steps within a certain use case, and link at the same time the exchange requirements and data validation shapes (e.g. using mvdXML or SHACL) to the different modules. With this in mind, we hope to come one step closer to a web-based, interdisciplinary and interoperable building industry.

5. Acknowledgements

The authors would like to acknowledge the support by both the BIM4REN project, which has received funding from the European Union's Horizon 2020 Research and Innovation Program under grant agreement No 820773 and the UGent Special Research Fund (BOF).

6. References

Afsari, K., Eastman, C.M., Shelden, D.R., 2016. Cloud-based BIM data transmission: current status and challenges, in: ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction. Vilnius Gediminas Technical University, Department of Construction Economics, p. 1.

Beetz, J., de Laat, R., van Berlo, L., Helm, P. van den, 2010a. Towards an Open Building Information Model Server, in: Proc. of the 10th International Conference on Design & Decision Support Systems in Architecture and Urban Planning.

Beetz, J., van Berlo, L., de Laat, R., van den Helm, P., 2010b. BIMserver. org–An open source IFC model server, in: Proc. of the CIP W78 Conference.

Beetz, J., Van Leeuwen, J.P., De Vries, B., 2005. An ontology web language notation of the industry foundation classes, in: Proceedings of the 22nd CIB W78 Conference. Technical University of Dresden, p. 670.

Berners-Lee, T., Hendler, J., Lassila, O., 2001. The semantic web. Sci. Am. 284, 34–43.

BLIS-project, 2002. BLIS Building Lifecycle Interoperable Software. http://www.blis-project.org/index2.html.

Bonduel, M., Oraskari, J., Pauwels, P., Vergauwen, M., Klein, R., 2018. The IFC to Linked Building Data Converter-Current Status, in: 6th Linked Data in Architecture and Construction Workshop. pp. 34–43.

Facebook Inc., 2016. GraphQL. Working Draft. https://graphql.github.io/graphql-spec/draft/

Hietanen, J., 2002. BLIS Review: IMSvr. BLIS. http://www.blis-project.org/

Jørgensen, K.A., Skauge, J., Christiansson, P., Svidt, K., Sørensen, K.B., Mitchell, J., 2008. Use of IFC Model Servers Modelling. Aalborg University.

Kiviniemi, A., Fischer, M., Bazjanac, V., 2005. Integration of multiple product models: IFC model servers as a potential solution, in: Proc. of the 22nd CIB-W78 Conference on Information Technology in Construction.

Mansour, E., Sambra, A.V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A., Berners-Lee, T., 2016. A Demonstration of the Solid Platform for Social Web Applications, in: Proc. of the 25th International Conference Companion on World Wide Web. International World Wide Web Conferences Steering Committee, pp. 223–226.

OASIS Open Project, n.d. OSLC - Open Services for Lifecycle Collaboration. URL https://open-services.net/

Pauwels, P., Terkaj, W., 2016. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. Autom. Constr. 63, 100–133.

Pauwels, P., Zhang, S., Lee, Y.-C., 2017. Semantic web technologies in AEC industry: A literature overview. Autom. Constr. 73, 145–165.

Rasmussen, M., Pauwels, P., Lefrançois, M., Schneider, G.F., Hviid, C., Karlshøj, J., 2017. Recent changes in the Building Topology Ontology, in: LDAC2017-5th Linked Data in Architecture and Construction Workshop.

Semantic Integration Ltd., n.d. HyperGraphQL. https://www.hypergraphql.org/

Shafiq, M.T., Matthews, J., Lockley, S., Love, P.E., 2018. Model server enabled management of collaborative changes in building information models. Front. Eng. Manag. 5, 298–306.

Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R., 2018a. Comunica: a modular SPARQL query engine for the web, in: International Semantic Web Conference. Springer, pp. 239–255.

Taelman, R., Vander Sande, M., Verborgh, R., 2019. Bridges between GraphQL and RDF.

Taelman, R., Vander Sande, M., Verborgh, R., 2018b. GraphQL-LD: Linked Data Querying with GraphQL, in: ISWC2018, the 17th International Semantic Web Conference.

van Berlo, L., 2013. BIM Service interface exchange (BIMSie). https://www.nibs.org/page/bsa_bimsie

van Berlo, L., n.d. bimbots.org - Automate your work with BIM Bots. http://bimbots.org.

Verborgh, R., 2018. Designing a Linked Data Developer Experience. https://ruben.verborgh.org/blog/2018/12/28/designing-a-linked-data-developer-experience/